יכולות שיקוף

הקדמה

השימוש ב-reflection לניתוח מחלקות

השימוש ב-reflection לניתוח אובייקטים בזמן ריצה

הקדמה

reflection היא היכולת לזהות, ולנתח משתנים, מתודות ו-constructors שמוגדרים במחלקות, וכמו כן, היכולת לזהות, לנתח ולהפעיל משתנים ומתודות של אובייקט מסוים בזמן ריצה. תכנית אשר מסוגלת לזהות ולנתח מחלקות/אובייקטים אחרות/אחרים בזמן ריצה מבלי לעשות שימוש בקובץ קוד המקור נקראת בשם: reflective program.

פעולות ה-reflection מבוצעות באמצעות המחלקה Class ובאמצעות המחלקות Field, Method, Array, Constructor, Modifier ו-AccessibleObject.

באמצעות reflection (וכאשר אין כל מגבלת security כמובן) ניתן לבצע את הפעולות הבאות:

לנתח ולמצוא את תכונותיהן של מחלקות (מתודות ומשתנים)

לבדוק (וגם לשנות/להפעיל) את תכונותיהם של אובייקטים (מתודות ומשתנים) בזמן ריצה

לייצור מתודה שתשמש להקטנתם/הגדלתם של מערכים קיימים

להפעיל בזמן ריצה מתודות באמצעות יצירת אובייקטים מטיפוס המחלקה Method (בדומה למצביעים לפונקציות ב- C++).

השימוש ביכולות ה- reflection בולט במיוחד בסביבות פיתוח ויזואליות שמאפשרות פיתוח יישומי Java בדומה לפיתוח יישומי Visual Basic.

השימוש ב-reflection לניתוח מחלקות קיימות

באמצעות המחלקה Class (זהו שם של מחלקה) ניתן לתאר מחלקה. אובייקט מטיפוס המחלקה Class מתאר מחלקה שהוגדרה. קיימות מספר דרכים לקבלת reference לאובייקט מטיפוס המחלקה Class אשר מתאר מחלקה מסוימת:

במחלקה Object מוגדרת המתודה getClass אשר מחזירה reference לאובייקט מטיפוס המחלקה Class אשר מתאר את המחלקה שאליה האובייקט שייך. ב-JAVA כל מחלקה שמגדירים (או שכבר הוגדרה לפני כן) יורשת באופן ישיר או עקיף מהמחלקה Object, ולכן מתודה זו ניתנת להפעלה על כל אובייקט.

דוגמא:

```
Car myCar = new Car();

Class classObj = myCar.getClass();
```

כעת, בתוך המשתנה classObj ישנו reference לאובייקט מטיפוס Class אשר מתאר את המחלקה Car.

במחלקה Class מוגדרת המתודה הסטטית forName אשר מקבלת בעת הפעלתה מחרוזת תווים שהיא שמה של מחלקה שהוגדרה וכתגובה מחזירה reference לאובייקט מטיפוס Class אשר מתאר את אותה מחלקה.

דוגמא:

```
Class classObj = Class.forName("Car");
```

כעת, בתוך המשתנה classObj ישנו reference לאובייקט מטיפוס Class אשר מתאר את המחלקה Car.

באמצעות הוספת המילה class אל סופו של שם של מחלקה (בתוספת של נקודה מפרידה ביניהם) ניתן לקבל reference לאובייקט מטיפוס המחלקה Class אשר מתאר את אותה מחלקה מסוימת.

דוגמא:

```
Class classObj = Car.class;
```

כעת, בתוך המשתנה classObj ישנו reference לאובייקט מטיפוס Class אשר מתאר את המחלקה Car.

באמצעות המחלקות:

`Field`

`Method`

`Constructor`

ששייכות ל-package ששמו `java.lang.reflect` ניתן לתאר את מרכיביהן של מחלקות. אובייקט מטיפוס כל אחת משלושת המחלקות האלה מתאר שדה (משתנה), מתודה ו-constructor בהתאמה. בכל אחת מהמחלקות האלה קיימות מתודות מתאימות שמאפשרות לקבל אינפורמציה על המרכיב המתואר.

כך, למשל, בכל אחת משלושת המחלקות קיימות המתודות הבאות:

```
String getName()
```

מתודה אשר מחזירה reference לאובייקט מטיפוס String אשר מתאר את שמו של המרכיב שמייצג האובייקט (שמו של השדה, המתודה או ה-constructor בהתאם).

```
Class getDeclaringClass()
```

מתודה אשר מחזירה reference לאובייקט מטיפוס Class אשר מתאר את המחלקה שאליה המרכיב שייך.

```
int getModifiers()
```

מתודה זו מחזירה ערך מטיפוס int, אשר באמצעות כל אחד מהביטים שלו ניתן לדעת את קיומו/אי קיומו של כל אחד מה-modifiers האפשריים (static, public, private וכדומה. . .). באמצעות המתודות הסטטיות המתאימות שקיימות במחלקה Modifier (isPrivate, isPublic, isFinal וכו'. . .) אשר פועלות על ערך ה-int אשר הוחזר על ידי המתודה

getModifiers ניתן לדעת את ערכו של כל אחד מה-modifiers במרכיב המתואר.

במחלקות Method ו- Constructor קיימות גם המתודות הבאות:

```
Class[] getExceptionTypes()
```

מתודה זו מחזירה reference למערך מטיפוס Class אשר רכיביו הנם references לאובייקטים מטיפוס Class שמתארים, כל אחד מהם, את הטיפוס של כל אחד מה-Exceptions שיכולים להיזרק על ידי המתודה. אם נושא ה-Exceptions (הטיפול בשגיאות) אינו מוכר לך, תוכל לקבל הבהרה לנושא בפרק שעוסק בטיפול בשגיאות.

```
Class[] getParameterTypes()
```

מתודה זו מחזירה reference למערך מטיפוס Class אשר רכיביו הנם references לאובייקטים מטיפוס Class שמתארים, כל אחד מהם, את הטיפוס של כל אחד מהפרמטרים של המתודה/constructor.

במחלקה Field קיימת המתודה הבאה:

```
Class getType()
```

מתודה זו מחזירה reference לאובייקט מטיפוס Class אשר מתאר את הטיפוס של המשתנה שמיוצג על ידי האובייקט.

במחלקה Method קיימת המתודה הבאה:

```
Class getReturnType()
```

מתודה זו מחזירה reference לאובייקט מטיפוס Class אשר מתאר את טיפוס הערך המוחזר על ידי המתודה.

זוהי רק סקירה מדגמית של הפעולות העיקריות שניתן לבצע באמצעות שלוש המחלקות הנ"ל. לקבלת אינפורמציה נוספת על המתודות שקיימות במחלקות אלה כדאי לפנות ל- API Documentation.

כל מחלקה קיימת ניתנת לתיאור באמצעות אובייקט שיוצרים מהמחלקה Class. אחת הדרכים שהוצגו בתחילת הפרק לקבלת reference לאובייקט מטיפוס Class אשר מתאר את המחלקה שאליה אובייקט מסוים שייך היא לעשות שימוש

במתודה `getClass` שניתנת להפעלה על כל אובייקט (מתודה זו מוגדרת במחלקה Object). מתודה זו מחזירה reference לאובייקט מטיפוס Class שמתאר את המחלקה שממנה נוצר האובייקט. המתודות השונות שמוגדרות במחלקה Class מאפשרות לקבל אינפורמציה על המחלקה המתוארת:

`Field [] getFields()`

מתודה זו מחזירה reference למערך מטיפוס המחלקה Field, אשר מתאר את כל השדות עם הרשאת הגישה public שקיימים במחלקה המתוארת.

`Method [] getMethods()`

מתודה זו מחזירה reference למערך מטיפוס Method אשר מתאר כל המתודות עם הרשאת הגישה public שקיימות במחלקה.

`Constructor [] getConstructors()`

מתודה זו מחזירה reference למערך מטיפוס Constructor אשר מתאר את כל ה-constructors עם הרשאת הגישה public, שקיימים במחלקה.

`Field [] getDeclaredField()`

מתודה זו מחזירה reference למערך מטיפוס Field אשר מתאר את כל השדות (משתנים) שמוגדרים במחלקה.

`Method [] getDeclaredMethods()`

מתודה זו מחזירה reference למערך מטיפוס Method אשר מתאר את כל המתודות שמוגדרות במחלקה.

`Constructor [] getDeclaredConstructors()`

מתודה זו מחזירה reference למערך מטיפוס Constructor אשר מתאר את כל ה-constructors שמוגדרים במחלקה.

התכנית הבאה מהווה דוגמה לשימוש פשוט במחלקות ובמתודות שנסקרו. כדי להפעילה עליך להוסיף לשורת ההפעלה (בסופה) שם של מחלקה כלשהי. כך, למשל, ניתן להריץ את התכנית באופן הבא:

```
java  ReflectionDemo  ReflectionDemo.java
```

ולקבל כפלט תיאור של כל המשתנים, המתודות וה-constructors שמוגדרים במחלקה ReflectionDemo. כדאי לשים

לב למתודה printSuperClasses אשר מדפיסה את שמות כל המחלקות שמורישות למחלקה האמורה באופן רקורסיבי.

אם אינך מכיר את נושא הרקורסיה אל תתעמק בהבנת המתודה בשלב זה.

```
//filename:ReflectionDemo.java

import java.lang.reflect.*;

public class ReflectionDemo

{

      private static String name;

      private static int blanks;

      public static void main(String[] args)

      {

            if (args.length!=1)

            {

                  System.out.println(

                 "\nERROR: You should give a name of ONE class !");

                  return;

            }



            name = args[0];

            Class classObj;

            try

            {
```

```
            classObj = Class.forName(name);
            System.out.println("The following report regards "
             + name + " : \n");

            if (!printSuperClasses(classObj))
                  System.out.println("This classObj doesn't"
                 + " extend any other classObj !");

            printConstructors(classObj);
            printMethods(classObj);
            printFields(classObj);
      }
      catch(ClassNotFoundException e)
      {
            System.out.println("Class not found.");
      }
}

public static void printBlanks()
{
      for (int i=0; i< blanks; i++)
            System.out.print(" ");
```

```
}



public static boolean printSuperClasses(Class classObj)
{
      if (classObj==null)
      {
            return false;
      }
      else
      {
            printSuperClasses(classObj.getSuperclass());
            System.out.println(classObj.getName());
            if (!classObj.getName().equals(name))
            {
                  printBlanks();
                  System.out.println("|");
                  printBlanks();
                  System.out.print("+--");
                  blanks+=(classObj.getName().length());
            }
            return true;
```

```
        }
    }

    public static void printConstructors(Class classObj)
    {
        Constructor[] constructors =
         classObj.getDeclaredConstructors();
        System.out.println("\nCONSTRUCTORS");
        System.out.println("============");
        for (int i = 0; i < constructors.length; i++)
        {
            System.out.println("constructor num." + (i+1));
            System.out.println("-----------------");
            Constructor c = constructors[i];
        Class[] paramTypes = c.getParameterTypes();
        String constructorName = c.getName();
        System.out.print(Modifier.toString(c.getModifiers()));
        System.out.print(constructorName + "(");
        for (int j = 0; j < paramTypes.length; j++)
        {
                if (j > 0) System.out.print(" , ");
                System.out.print(paramTypes[j].getName());
```

```
        }
        System.out.println(");");
        }
}

public static void printMethods(Class classObj)
{
        Method[] methods = classObj.getDeclaredMethods();
        System.out.println("\nMETHODS");
        System.out.println("=======");
        for (int i = 0; i < methods.length; i++)
        {
                System.out.println("method num." + (i+1));
                System.out.println("------------");
                Method m = methods[i];
                 Class retType = m.getReturnType();
                Class[] paramTypes = m.getParameterTypes();
                String methodName = m.getName();
               System.out.print(
                 Modifier.toString(m.getModifiers()));
                System.out.print(retType.getName() + " " +
                 methodName + "(");
```

```
            for (int j = 0; j < paramTypes.length; j++)
             {
                  if (j > 0) System.out.print(", ");
                    System.out.print(paramTypes[j].getName());
             }
            System.out.println(");");
      }
}


public static void printFields(Class cl)
{
      Field[] fields = cl.getDeclaredFields();


      System.out.println("\nFIELDS");
      System.out.println("======");
      for (int i = 0; i < fields.length; i++)
      {
            System.out.println("field num." + (i+1));
            System.out.println("-----------");
            Field f = fields[i];
            Class type = f.getType();
            String fieldName = f.getName();
```

```
                System.out.print(
                  Modifier.toString(f.getModifiers()));
                System.out.println(" " + type.getName() + " "
                  + fieldName + ";");
            }
        }
    }
```

השימוש ב-reflection לניתוח אובייקטים בזמן ריצה

פניה לערכיהם של משתנים באובייקט מסוים כאשר קיומו ידוע בשלב ההידור איננה דבר מסובך, וכבר ראית זאת בפרק שדן במחלקות. עם זאת, שימוש בדרכים המקובלות כדי לפנות לערכיהם של משתנים באובייקט שנוצר בזמן פעולת התכנית, ושבשלב ההידור הטיפוס שלו עדיין לא היה ידוע, לא תמיד אפשרי.

באמצעות המחלקות שנסקרו בחלק הקודם של הפרק ניתן לבחון את הערכים במשתנים שיש באובייקט מסוים גם כאשר קיומו וטיפוסו של האובייקט לא ידועים בשלב ההידור. באופן דומה, באמצעות המחלקות האמורות ניתן גם לגלות בזמן ריצה מהן המתודות שניתן להפעיל על אובייקט מטיפוס שבזמן כתיבת הקוד לא היה ידוע.

על כל אובייקט ב-JAVA ניתן להפעיל את המתודה `getClass` ולקבל בחזרה reference לאובייקט מטיפוס `Class` אשר מתאר את המחלקה שממנה נוצר אותו אובייקט. באמצעות ה-reference שמתקבל ניתן להפעיל מתודות שונות (חלק מהן נסקרו בחלק הקודם של הפרק) כדי לקבל אינפורמציה על רכיבי אותה מחלקה שממנה נוצר האובייקט, כגון: getFields(), getDeclaredFields, getMethods(), getDeclaredMethods(), getConstructors(), getDeclaredConstructors(). כל אחת מהמתודות האלה מחזירה reference למערך של אובייקטים אשר מתארים את המתודות, המשתנים ואת ה-constructors. מערכים אלה כוללים references לאובייקטים מטיפוס Field/Method/Con structor בהתאם, ובאמצעותם ניתן להפעיל את המתודות השונות שמוגדרות במחלקות Field/Method/Constructor. בדרך זו ניתן, למשל, להפעיל על כל אחד מן האובייקטים מטיפוס Field (אובייקטים אשר מתארים את המשתנים שיש באובייקט) את המתודה get אשר מקבלת reference לאובייקט שמשתנה זה קיים בו, ומחזירה את ערכו של אותו משתנה באובייקט.

לעתים, כאשר מנסים לבדוק את ערכו של משתנה באובייקט מסוים, והרשאת הגישה לאותו משתנה private, הגישה לקבלת ערכו לא תתאפשר. במקרים כגון אלה, ובהנחה שה- security manager מאפשר זאת, ניתן להפעיל על אובייקט ה-`Field` את המתודה `setAccessible` ולשלוח אליה true. מתודה זו מגיעה אל המחלקות `Method`,

Constructor ו- Field בירושה מהמחלקה AccessibleObject. בתכנית לדוגמא שמופיעה בהמשך ניתן לראות הדגמה להפעלתה של מתודה אחרת ששייכת גם היא למחלקה AccessibleObject, שמקבלת reference למערך של references מטיפוס Field וערך מטיפוס boolean, ומאפשרת/לא מאפשרת את הגישה לכל אחד מהשדות עפ"י הערך הבוליאני שקיבלה.

כאשר מדובר במשתנה מחלקתי מטיפוס בסיסי (Primitive Type) אשר מתואר על ידי אובייקט מטיפוס Field ניתן באמצעות המתודה get (אשר מוגדרת במחלקה Field) לקבל בחזרה reference לאובייקט מטיפוס המחלקה המתאימה מבין ה- wrapper classes (קבוצת המחלקות אשר מתארות את כל אחד מהטיפוסים הבסיסיים). לדוגמא, אם מדובר במשתנה (Field) מטיפוס int, אז המתודה get תחזיר reference לאובייקט מטיפוס Integer (המחלקה Integer היא wrapper class אשר מייצגת את הערך int).

בתכנית הבאה, תוצג הדגמה למה שנאמר עד כה בנוגע לבחינת ערכיו של אובייקט בזמן ריצה.

```
//filename:ObjectInspector.java


import java.lang.reflect.*;


public class ObjectInspector

{

      Object obj;

      Class classObj;

      public ObjectInspector(Object obj)

      {
```

```
        this.obj = obj;
        classObj = obj.getClass();
    }

    public void printClassName()
    {
        System.out.println(classObj.getName());
    }

    public void printFieldValues()
    {
        Field [] fields = classObj.getDeclaredFields();
        try
        {
            AccessibleObject.setAccessible(fields, true);
        }
        catch(SecurityException e)
        {
            e.printStackTrace();
        }
        for(int i=0; i<fields.length; i++)
        {
```

```
            System.out.println("field num." + (i+1));
            System.out.println("-----------");
            System.out.println("modifiers : " +
              Modifier.toString(fields[i].getModifiers()));
            System.out.println("type : " +
                fields[i].getType());
            System.out.println("name : " +
                fields[i].getName());
            try
            {
                  System.out.println("value : " +
                     fields[i].get(obj));
            }
            catch(Exception e)
            {
                  e.printStackTrace();
            }
      }
}

public static void main (String args[])
{
```

```
            Punto pun = new Punto(3,4);

            ObjectInspector objIns = new ObjectInspector(pun);

            System.out.println("The class of pun :");

            objIns.printClassName();

            System.out.println("The fields of pun :");

            objIns.printFieldValues();

      }

}




class Punto

{

      private int x, y;



      public Punto(int a, int b)

      {

            x=a;

            y=b;

      }

}
```

באמצעות המתודה set שמוגדרת במחלקה `Field` ניתן גם לקבוע את ערכיו של אובייקט בזמן ריצה. לדוגמא, אם נתון אובייקט מטיפוס `Punto` שה-referemce אליו מאוחסן במשתנה ששמו pun, ובמשתנה f (משתנה מטיפוס `Field`) מוחזק reference לאובייקט מטיפוס Field שמייצג את אחד ממשתניו, אז ניתן באמצעות הפעלת המתודה set ושליחת הערך המתאים (עפ"י הטיפוס) לקבוע את הערך של אותו משתנה (שמיוצג על ידי אובייקט ה-`Field`) באובייקט pun.הערך הראשון ששולחים למתודה set הוא reference לאובייקט שבו רוצים לקבוע ערך חדש למשתנה, והערך השני ששולחים הוא הערך שרוצים שיוכנס לתוך אותו משתנה.

```
f.set(pun, 72);
```

שורה זו, למשל, תגרום לכך שערכו של המשתנה x באובייקט שהמשתנה pun מחזיק ב-reference אליו יהיה 72.